Using and Re-using Application Generators

A Plantec, Vincent Ribaud

Using and Re-using Application Generators

A.Plantec and V.Ribaud
Syseca & LIBr
Syseca: 34 quai de la Douane, 29200 Brest, France
LIBr: Faculté des sciences, Département d'Informatique, 29285 Brest Cedex, France
E-mail:{plantec,ribaud}@univ-brest.fr

Abstract

Application generators translate specifications into products (programs, documentation). They parse specification statements into data structures (called dictionaries or meta-models), from which desired products can be derived. An application generator builder offers a way to define specification languages and associated parsers, to describe and traverse the meta-models structure and to specify the derivation on this structure.

STEP is an ISO 10303 standard developed to facilitate product information sharing by specifying sufficient semantic content for data and their usage. STEP offers very useful technology, such as an object oriented modelling language EXPRESS and a data access interface, and can be applied for the design and the implementation of application generators.

EUGENE is a STEP-based framework, that is intended for the building of application generators.

After a concise presentation of EUGENE, this paper presents different application generators built with this framework, mainly data management tools, a browsing tool and conversion tools. Then, we discuss the main benefits induced from our approach.

1. Introduction

Application generators translate source specifications into target products as programs or documentation. They use data structures called dictionaries or meta-models to store the source specifications and intermediate representations. As in compilers, there are two main functions in a generator: parsing of the source specifications and the target code generation.

Typically, an application generator builder offers a means to define specification languages and associated parsers, to describe and traverse the meta-model structure and to specify the derivation on this structure. In most builders, derivation is specified with templates (or skeletons) of code. Templates contain a mixture of commands operating on the meta-models and "real" code directly inserted into the products. With classical tools and methods, specifying and implementing a generator is very much like specifying and implementing a language.

In this paper, we describe a way of building application generators directly from the object-oriented specification of meta-models. We used this possibility in a tool, called EUGENE [8]. Meta-models and code templates are programmed with the object-oriented data modelling langage, EXPRESS.

We argue that building generators from the specification of meta-models can simplify their design and implementation and makes their use and re-use in industrial projects easier. As a consequence, generators can be implemented for specific problems and generated realizations can be well integrated.

EUGENE is briefly presented in chapter 2. In chapter 3, we show different generators built with EUGENE. We discuss the main benefits of EUGENE in chapter 4.

2. Building application generators with EUGENE

An application generator built with EUGENE uses meta-data in order to generate a target textual representation. Meta-data come from a source specification analysis and are stored in different kind of meta-models. Derivations are programmed with imperative functions, called translation functions. They are made of fixed parts that are mainly either meta-model traversal routines or string constants directly put into the target and made of variable parts that are values fetched from the meta-models.

Figure 1 shows that an application generator built with EUGENE is only a process that consumes meta-data and produces some realization. The meta-data are themselves produced by other processes or tools. Meta-data consist of a representation of the source specification called *intermediate representation*. With EUGENE, any meta-data producer can be used. Most application generators builders produce the generators while exploiting the definition of the syntax and the semantic of the source specification language [2, 3, 4]. EUGENE differs from this way, because the generator is not produced from a grammar-like specification, but from the structure and the organization of the meta-data (globally referred to as the meta-models).

STEP description and implementation methods

The EXPRESS language [1] is an object-oriented modelling language. The application data are described in schemata. A schema has the type definitions and the object descriptions of the application called Entities. An entity is made up of attributes and constraint descriptions.

The constraints expressed in an entity definition can be of four kinds: (1) the unique constraint allows entity attributes to be constrained to be unique either solely or jointly, (2) the derive clause is used to represent computed attributes, (3) the where clause of an entity constraints each instance of an entity individually and (4) the inverse clause is used to specify the inverse cardinality constraints. Entities may inherit attributes and constraints from their supertypes.

The STEP physical file format defines an exchange structure using a clear text encoding of product data for which a conceptual model is specified in the EXPRESS language. The mapping from the EXPRESS language to the syntax of the exchange structure is specified in [2].

The Standard Data Access Interface (SDAI) [3] defines an access protocol for EXPRESS-modelled databases and is defined independently from any particular system and language. The representation of this functional interface in a particular programming language is referred to as a language binding in the standard. As an example, ISO 10303-23 is the STEP part describing the C++ SDAI binding [4].

The five main goals of the SDAI are: (1) to access and manipulate data which are described using the EXPRESS language, (2) to allow access to multiple data repositories by a single application at the same time, (3) to allow commit and rollback on a set of SDAI operations, (4) to allow access to the EXPRESS definition of all data elements that can be manipulated by an application process, and (5) to allow the validation of the constraints defined in EXPRESS.

An SDAI can be implemented as an interpretor of EXPRESS schemata or as a specialized data interface. The interpretor implementation is referred to in the standard [3] as the SDAI late binding. An SDAI late binding is generic in nature. The specialized implementation is referred to in the standard as the SDAI early binding.

References

[1] ISO 10303-11. Part 11 : EXPRESS Language Reference Manual, 1994.
[2] ISO 10303-21. Part 21 : Clear Text Encoding of the Exchange Structure, 1994.
[3] ISO DIS 10303-22. Part 22 : Standard Data Access Interface, 1994.
[4] ISO CD 10303-23. Part 23 : C++ Programming Language Binding to the SDAI Specification, 1995.

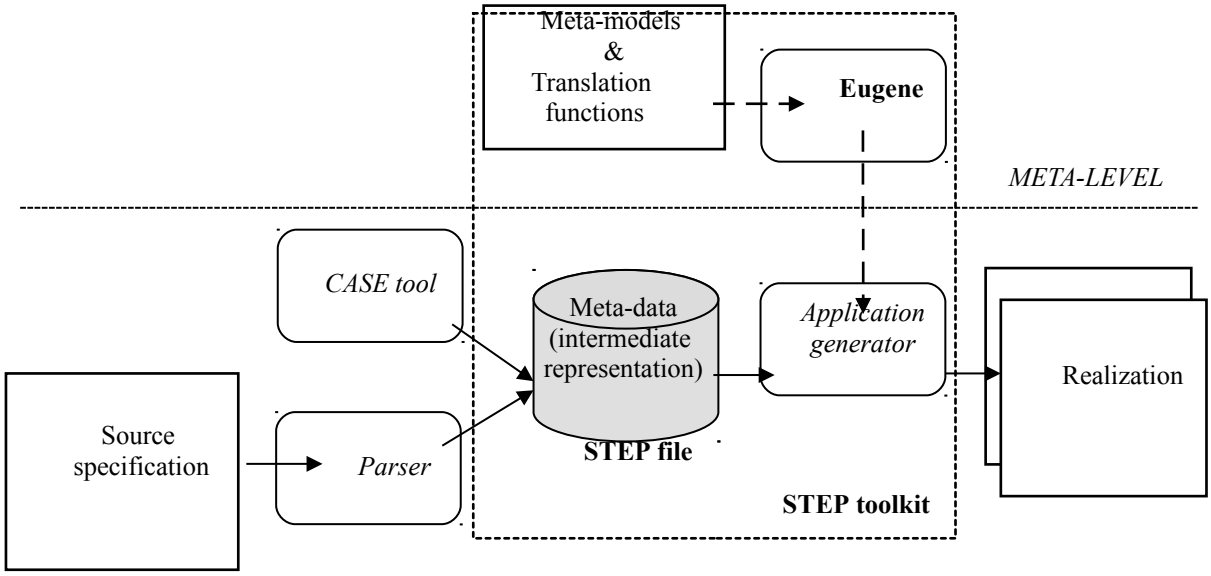

Figure 1 The building and the using of an application generator

Thus the tool which feeds the meta-models with meta- data can be a classical parser (generally automatically

built in other environments than EUGENE) but another tool not relying on a precise and formal concrete syntax.

Let us take the example of the classical work done by a screen generator of a Relational Data Base Management System (RDBMS). The generator uses the SQL table definitions (usually stored in a dictionary) and produces a graphical form (a window with fields, buttons,etc.) for a given user interface toolkit (Windows, X, etc.).

2.1 The meta-models

The specification of the generator being built consists in several meta-models. A precise explanation of the nature of these meta-models can be found in [8]. To simplify the example, we will use one source meta-model.

The source language meta-model consists in a set of EXPRESS schemata that describes the source language data constructs. The main components of a source language meta-model are types and entities, describing concepts that can be used with the source language. Entities provide buckets to store meta-data while global and local EXPRESS constraints are used to ensure meta-data soundness.

Considering the classical example of building form screens from SQL table definitions, the source language is SQL and figure 2 shows a simplified SQL meta-model. The table definition is related to a list of columns.

```
SCHEMA sql_dictionary;

  ENTITY simple_type  SUPERTYPE
     OF (ONEOF(real_type, integer_type, string_type));
  END_ENTITY;  ...

  ENTITY table;
     name : STRING;
     columns : LIST [1:?] OF column; ...
  END_ENTITY;

  ENTITY column;
     name : STRING;
     domain : simple_type;
  END_ENTITY; ...
END_SCHEMA ;
```

Figure 2 An example of source meta-model: a simple SQL dictionary

2.2 The translation functions

The translation functions are also written in EXPRESS and are specified in the translation schema. The specification of translation functions is a programming activity in which EXPRESS is used as an imperative language. A typical translation function returns a string and is parameterized with types that are entities defined in the source language meta-model. The resulting string represents part of the target textual representation. Because of the nature of parameter types, this activity is often called meta-programming [1, 5].

Figure 3 shows two skeletons of translation functions related to the paper example. *column_to_field* and *table_to_form* can be used in order to produce the definitions of widgets for, respectively, a column and a table.

```
SCHEMA sql_traduction;
 USE FROM sql_dictionary ;

  FUNCTION column_to_field ( c : column) : STRING ;
  LOCAL
    result : STRING := '';
  END_LOCAL ;
    (* build statements for the management of a
     * column in a graphical form*)
     RETURN (result) ;
  END_FUNCTION ;

  FUNCTION table_to_form( t : table ) : STRING ;
  LOCAL
    result : STRING := '';
  END_LOCAL ;
    (* build statements for the management of a
     * table in a graphical form *)
    REPEAT no := LOINDEX(t.columns) TO IINDEX(t.columns) ;
      ....
    END_REPEAT ;
    RETURN (result) ;
  END_FUNCTION ;

END_SCHEMA ;
```

Figure 3 An example of translation functions

3. Applications

As most application generator builders, EUGENE can be used to produce software components from source specifications of a high abstraction level.

Two conditions are a prerequisite, namely :

- the intermediate representation of the source specification can be described with an EXPRESS meta-schema;
- the realization can be produced with imperative processing applied to meta-data.

3.1 Data management tools

EUGENE was primarily intended to build data management components generators. The semantic of the domain is well-known (roughly query, insert, update and delete data). Data types are the variable part, they parametrize the realization. The way the implementation is carried out depends on data types and on the storage system (file, database, …) called a repository. Hence, generators for the data management domain are widely used.

Specifications for such generators are essentially data structures as records, classes, schemas. Realizations, built from these specifications, manage repository in a specific language such as C, SQL, C++. The SDAI is an abstraction of the data management domain. From the conceptual description (in EXPRESS) of the domain entities, the SDAI implementation is produced (the functionalities of the SDAI are shown in the box STEP

given above). Another kind of generator, called IO, provides the basic access interface to SQL tables through a host language; embedded SQL for query and update is generated from a data structure description.

As depicted in figure 1, a complete generator is made up of a parser and an application generator. The table below presents the realization time (in man-month) of several parsers (we re-used empty yacc-parsers):

Specification language	*Development time*
C structures	0.1
SQL schemas	0.5
EXPRESS schemas	1.5

We used EUGENE to build various generators, mainly SDAI generators. The next table presents the different generators built and the development time (in man-month) :

Source lang. / *Target lang.*	*C structure*	*SQL schema*	*EXPRESS*
C+SQL	0.5 (IO)		
C++			2 (SDAI)
Smalltalk			1 (SDAI)
Java			1 (SDAI)
Vis Basic+SQL		0.5 (IO)	0.5 (IO)

3.2 Browsing tool

EUGENE was used to build an instance browser generator (see figure 4). Given a data schema as specification source, a data management application in Java is generated. The generated application enables the user to browse the instances of a repository, to read and write the values associated with the instances and to navigate along their relationships. This instance browser generator produces two components : a graphical user interface and a SDAI to manage the instances. The second component is produced by the Java SDAI generator described in a previous paragraph. A three-layered generator produces the graphical user interface, each layer consuming meta-data produced by the former layer.

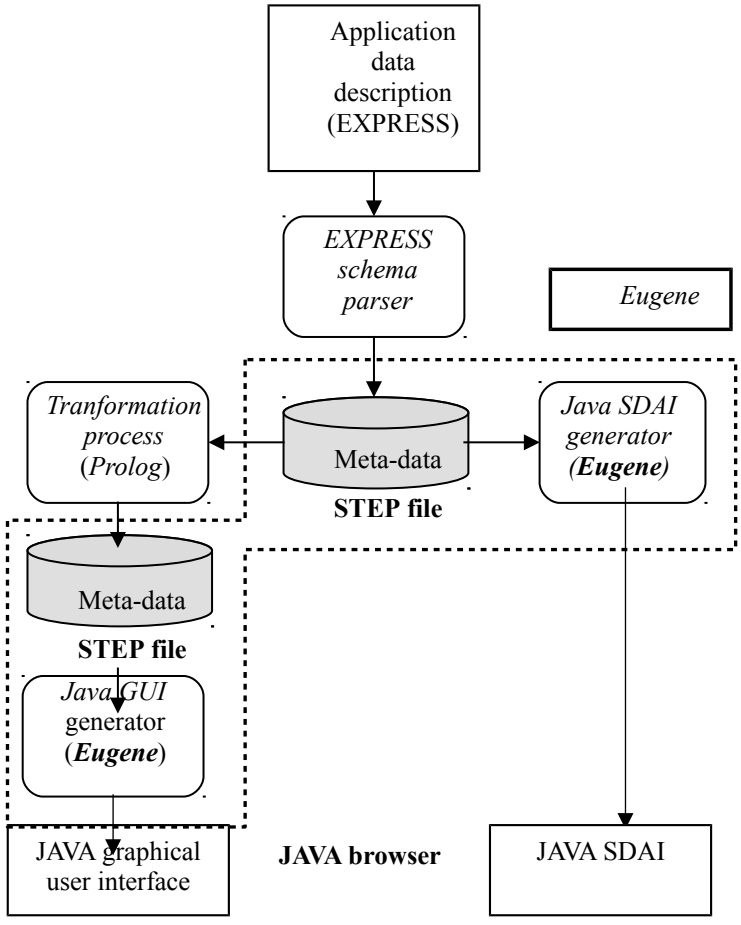

Figure 4 Java data browser generation

3.3 Conversion tools

A conversion tool transforms a source system

representation into a target system representation. Unlike former tools, a conversion tool has no leverage effect, i.e. the complexity of generated products is similar to the source representation.

Documentation tools are kinds of conversion tools. These tools use a meta-model which describes information extracted from source documents. This meta-model is used to reproduce the document according to the target representation.

This architecture was used in two tools: ADOC, a program documentation generator and an indexation and navigation tool for blind people.

ADOC produces Latex program documentation from the analysis of structured comments contained in programs sources. The comments should be structured according to a BNF grammar. ADOC uses a Lex/Yacc parser and a generator built with EUGENE. The meta-schema, derived from the BNF grammar, is populated by the parser. It is then used by the generator to produce Latex documents.

The second tool aims to make it easier for blind people to navigate within Web documents. Such documents are originally written using a word processor (e.g. Latex) and then generated in HTML using a converter (e.g. Latex2HTML). The tool consists in a HTML scanner and a HTML generator built with EUGENE. The tree of Web pages is scanned to construct a word index and the hierarchical organization in chapter, section, and so on. The scan result, in the form of a STEP file exchange, is consumed by the generator to produce a table of contents, a plan and a word index, all hyperlinked to the original tree. Rather than exploring in depth the Web tree, blind people consult these meta-structures prior to navigating in the Web pages.

The joint use of both tools provides meta-structures to navigate in program documentation. As depicted in figure 5, inter-operability between these two tools can be achieved in two ways : either by the Latex2HTML converter, or by a gateway built with EUGENE which populates the meta-schema used by the HTML generator from the meta-data resulting from the comments analysis.

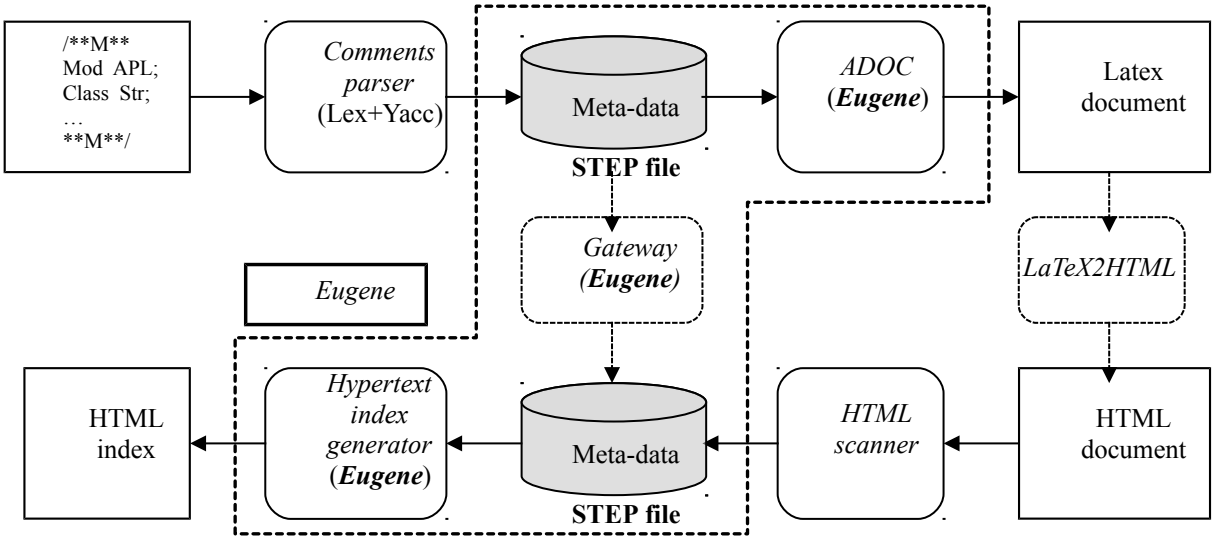

Figure 5 Conversion tools

4. Discussion

4.1 Usability

EUGENE uses an unique language, EXPRESS, for the meta-model specification and translation functions. Using EUGENE requires essentially data design and imperative programming. These activities are familiar to software engineers.

Parsers are not built with EUGENE, rather with well-known compiler-compiler such as Yacc. Meta-model implementation is automatically derived from specification. Parsers that have to be developed can use this implementation in order to create an intermediate representation.

4.2 Re-use

As mentioned in §3.1, EUGENE was used to build three SDAI-subset generators, for C++, Java and Smalltalk-80. While the specification source language is the same, the three SDAI-subset generators re-use the same parser and source meta-model. While the target languages (C++, Java, Smalltalk-80) are object-oriented, the translation functions of the C++ generator serve as a good example for the translation functions of the two

latter generators.

The HTML scanner of §3.3 was developed in a separate project. It was re-used without changes, adding only a feature intended to flush the scan results in a STEP file, while using the SDAI built for the meta-model of the HTML generator.

A design and validation method of data models is defined in the STEP standard [6]. EUGENE applies a part of this method in order to develop and re-use the meta-models.

4.3 Interoperability

Thanks to the STEP standard, meta-data can be automatically encoded in neutral ASCII files. It enables generators to read meta-data independently from their producers. Then a generator built with EUGENE is easily interoperable.

The instances browser depicted in § 3.2 is made up of two components, sharing the same meta-model (the same specification language), thus making it possible for the two generators to inter-operate. The overall project was implemented by different people over a period of three years, and thanks to the integration method no major problem has arisen.

In § 3.3, figure 5 shows that the hypertext index generator consumes meta-data produced by an HTML parser. In order to build HTML indexes for programs comments, a first solution consists in using ADOC generator and Latex2HTML. In order to re-use the hypertext index generator, a second solution could be to produce meta-data needed by this generator directly from the meta-data produced by the comments parser.

5. Conclusion

This paper has briefly described the EUGENE environment, a STEP-based application generator builder. A generator is automatically built from the specification of the intermediate representation manipulated by the generator (the meta-schema) and from the specification of translation functions.

Different generators are presented: data management tools, browsing tool, conversion tools.

Using EUGENE offers three main benefits :

1. *easy use*: half of projects had be done within the context of final-year coursework (postgraduate students) and the other half in a software compagny. All the projects succeeded in producing valid prototypes or operational generators.
2. *re-use*: several meta-schemata and software components (parsers or converters) were re-used between the different projects and are suitable for future re-use.
3. *inter-operability*: all the projects demonstrated that software integration through meta-data exchange is a suitable characteristic for both the integration of the generator into its environment and its re-use.

References

[1] Y. Ait-Ameur, F. Besnard, P. Girard, G. Pierra, and J. C. Potier. Formal Specification and Metaprogramming in the EXPRESS langage. In Int'Conf' on Software Engineering and Knowledge Engineering (SEKE), 1995.

[2] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: the system. In ACM SIGSOFT'88, Third annual symposium on software development environment, 1988.

[3] J. C. Cleaveland. Building Application Generators. IEEE Software, July 1988.

[4] Paul Klint. A Meta-Environment for Generating Programming Environments. In ACM Transaction on Software Engineering and Methodology, volume 2, 1993.

[5] David A. Ladd and J. Christopher Ramming. A*: A Language for Implementing Language Processors. IEEE Transactions on Software Engineering, 21(11), November 1995.

[6] M. Palmer. Guidelines for the development and approval of STEP application protocols. Technical report, ISO TC184/SC4/WG4 N511, 1995.

[7] Alain Plantec and Vincent Ribaud. EUGENE: a STEP-based framework to build Application Generators. AWCSET'98, CSIRO-Macquarie University, 1998.

[8] Alain Plantec. PhD thesis, Université de Rennes I, 35065 Rennes cedex, France, 11 Février 1999.